UNIFESP
UNIVERSIDADE FEDERAL DE SÃO PAULO
1933

Instituto de Ciência e Tecnologia

Universidade Federal de São Paulo

Compiladores:
A linguagem TINY e o gerador FLEX

Profª Thaína A. A. Tosta

tosta.thaina@unifesp.br

São José dos Campos – 2021/2

A linguagem TINY e o gerador FLEX

A linguagem TINY será usada como exemplo para ilustrar características essenciais de um compilador.

Ver seção 1.7 do livro "Compiladores: Princípios e Práticas", Kenneth C. Louden

A linguagem TINY e o gerador FLEX

Características gerais da linguagem TINY:

- Sequências de declarações separadas por ponto-e-vírgula (com exceção da declaração final);
- Variáveis são do tipo inteiro ou booleano;
- Variáveis declaradas pela atribuição de valores;
- Há duas declarações de controle: *if* e *repeat*;
- *If* tem como opcional uma parte *else*, e precisa terminar com a palavra-chave *end.*

A linguagem TINY e o gerador FLEX

Características gerais da linguagem TINY:

- Há declarações para entrada e saída:
  read
  write
- Permite comentários que aparecem entre chaves
  Não podem ser aninhados
- Permite expressões aritméticas com inteiros
  `+ - * /` (divisão inteira)
- Uma expressão aritmética pode ter constantes, variáveis e parênteses.

A linguagem TINY e o gerador FLEX

Características gerais da linguagem TINY:

- Permite expressões booleanas

  Expressão booleana composta por uma comparação de duas expressões aritméticas;

  Operadores relacionais: `<` e `=`.
- A linguagem não contempla:

  Funções e Procedimentos;

  Matrizes;

  Estruturas;

  Valores em ponto flutuante.

A linguagem TINY e o gerador FLEX

Exemplo de um programa em TINY

```
{ Sample program
  in TINY language -
  computes factorial
}
read x; { input an integer }
if 0 < x then { don't compute if x <= 0 }
  fact := 1;
  repeat
    fact := fact * x;
    x := x - 1
  until x = 0;
  write fact  { output factorial of x }
end
```

A linguagem TINY e o gerador FLEX

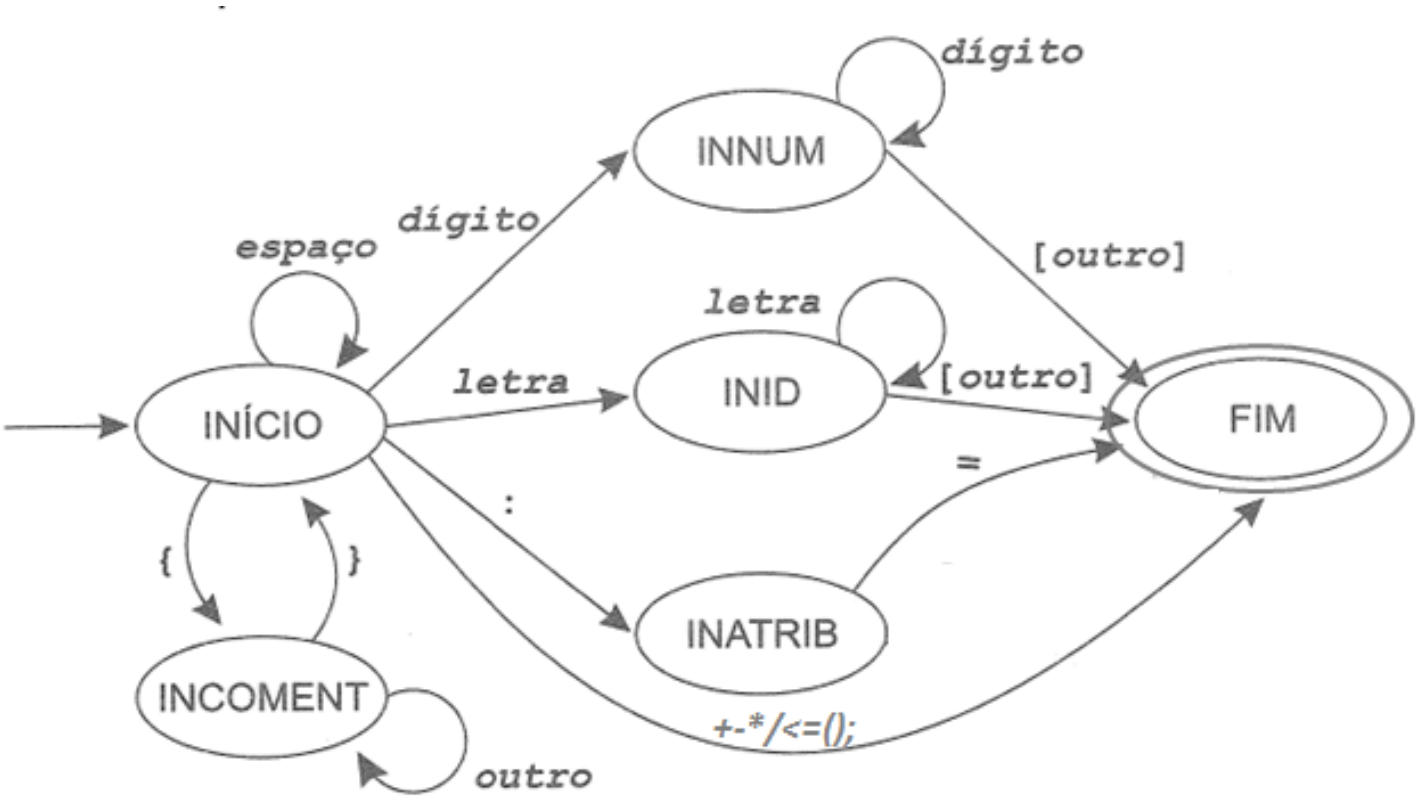
Estrutura léxica da linguagem TINY

Tabela 2.1 Marcas da linguagem TINY

Palavras Reservadas	Símbolos Especiais	Outras
`if`	`+`	*`número`*
`then`	`-`	(1 ou mais
`else`	`*`	dígitos)
`end`	`/`	
`repeat`	`=`	
`until`	`<`	*`identificador`*
`read`	`(`	(1 ou mais
`write`	`)`	letras)
	`;`	
	`:=`	

A linguagem TINY e o gerador FLEX

AFD para o analisador léxico de TINY



[outro] : não avança caractere

A linguagem TINY e o gerador FLEX

programa → *decl-seqüência*
decl-seqüência → *decl-seqüência* **;** *declaração* | *declaração*
declaração → *cond-decl* | *repet-decl* | *atrib-decl* | *leit-decl* | *escr-decl*
cond-decl → **if** *exp* **then** *decl-seqüência* **end**
| **if** *exp* **then** *decl-seqüência* **else** *decl-seqüência* **end**
repet-decl → **repeat** *decl-seqüência* **until** *exp*
atrib-decl → **identificador** **:=** *exp*
leit-decl → **read identificador**
escr-decl → **write** *exp*
exp → *exp-simples comp-op exp-simples* | *exp-simples*
comp-op → **<** | **=**
exp-simples → *exp-simples soma termo* | *termo*
soma → **+** | **-**
termo → *termo mult fator* | *fator*
mult - ***** | **/**
fator → **(***exp***)** | **número** | **identificador**

Figura 3.6 Gramática da linguagem TINY em BNF.

A linguagem TINY e o gerador FLEX

programa → *decl-seqüência*
decl-seqüência → *declaração* { **;** , *declaração* }
declaração → *if-decl* | *repeat-decl* | *atribuição-decl* | *read-decl* | *write-decl*
if-decl → **if** *exp* **then** *decl-seqüência* [**else** *decl-seqüência*] **end**
repeat-decl → **repeat** *decl-seqüência* **until** *exp*
atribuição-decl → **identificador** **:=** *exp*
read-decl → **read** **identificador**
write-decl → **write** *exp*
exp → *simples-exp* [*comparação-op simples-exp*]
comparação-op → **<** | **=**
simples-exp → *termo* { *soma termo* }
soma → **+** | **-**
termo → *fator* { *mult fator* }
mult → ***** | **/**
fator → **(** *exp* **)** | **número** | **identificador**

Figura 4.9 Gramática da linguagem TINY em EBNF.

A linguagem TINY e o gerador FLEX

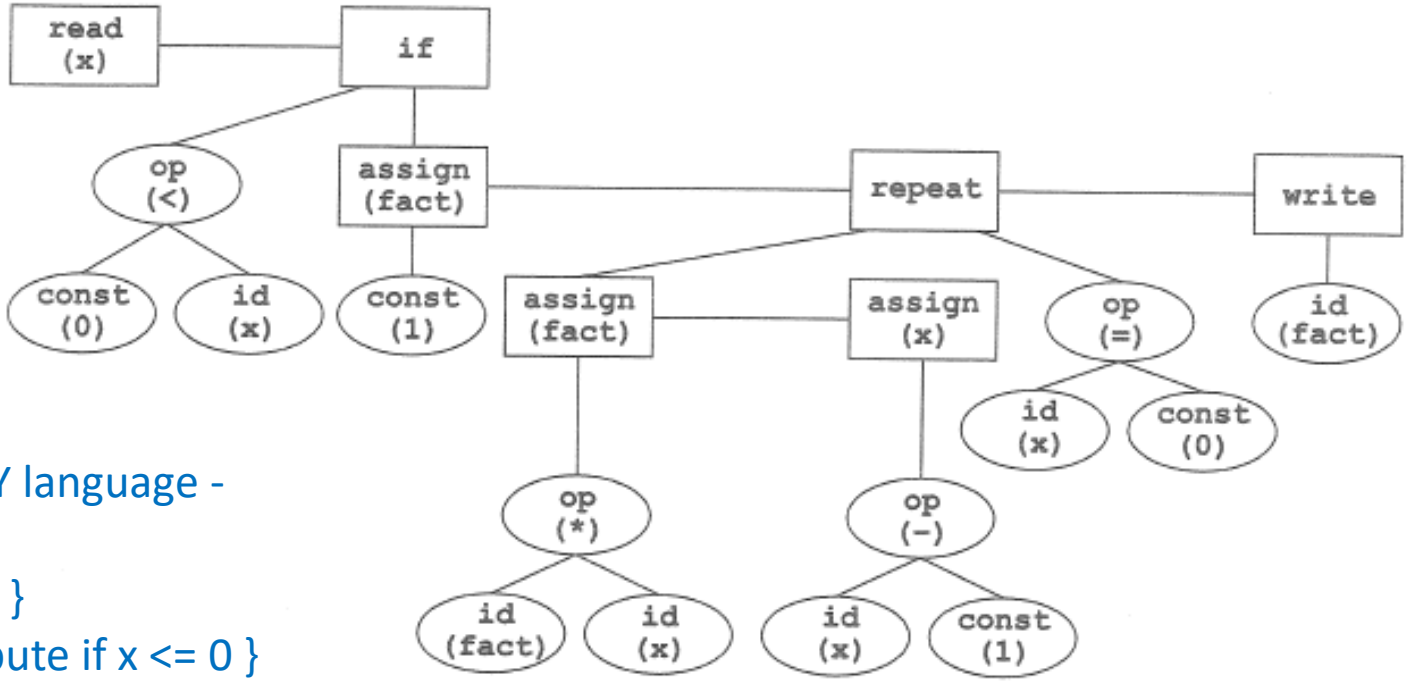


Figura 3.9 Árvore sintática para o programa TINY da Figura 3.8.

```
{ Sample program in TINY language -
  computes factorial }
read x; { input an integer }
if 0 < x then { don't compute if x <= 0 }
  fact := 1;
  repeat
    fact := fact * x;
    x := x - 1
  until x = 0;
  write fact  { output factorial of x }
end
```

A linguagem TINY e o gerador FLEX

```
typedef enum {StmtK,ExpK} NodeKind;
typedef enum {IfK,RepeatK,AssignK,ReadK,WriteK}
             StmtKind;
typedef enum {OpK,ConstK,IdK} ExpKind;

/* ExpType é utilizado para verificação de tipos */
typedef enum {Void,Integer,Boolean} ExpType;

#define MAXCHILDREN 3

typedef struct treeNode
   {  struct treeNode * child[MAXCHILDREN];
      struct treeNode * sibling;
      int lineno;
      NodeKind nodekind;
      union { StmtKind stmt; ExpKind exp;} kind;
      union { TokenType op;
              int val;
              char * name; } attr;
      ExpType type; /* para verificação de tipos de expressões */
   }  TreeNode;
```

Figura 3.7 Declarações em C para um nó de árvore sintática TINY.

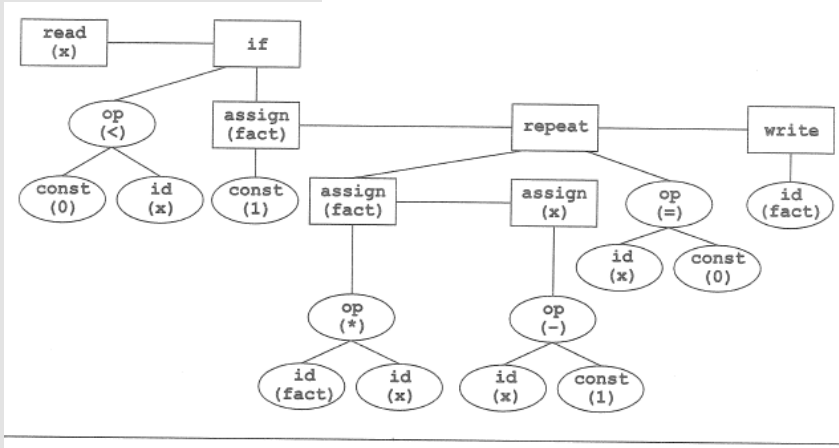


Figura 3.9 Árvore sintática para o programa TINY da Figura 3.8.

A linguagem TINY e o gerador FLEX

Atividade

Implementar um algoritmo (dirigido por tabela) que reconheça todos os *tokens* da linguagem *TINY*;

O programa deve ler o arquivo sample.tny e fornecer a resposta como a ilustrada ao lado.

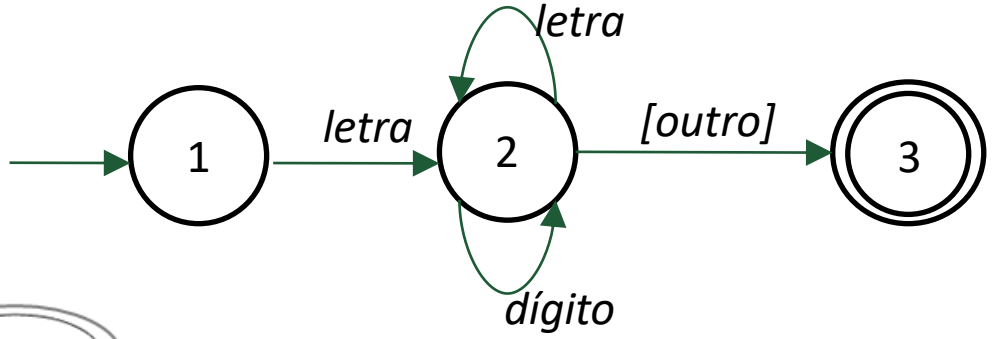
Inicialmente construir a tabela de transição de estados, de acordo com o exemplo visto em aula.

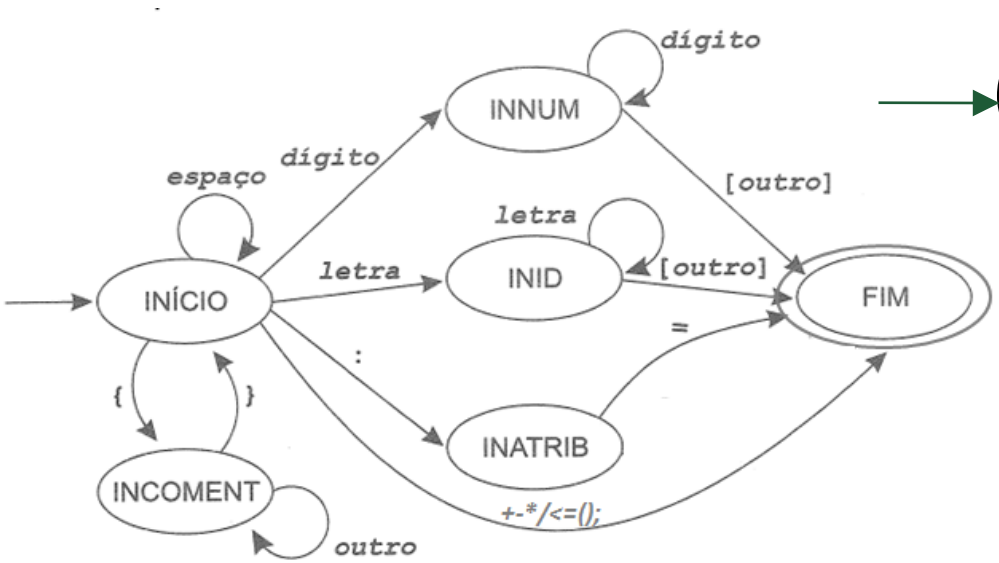
```
TINY COMPILATION: sample.tny
   1: {  Programa de exemplo
   2:     na linguagem TINY -
   3:     computa o fatorial
   4: }
   5: read x; {entrada de um inteiro}
      5: reserved word: read
      5: ID, name= x
      5: ;
   6: if 0 < x then {não calcula se x <= 0}
      6: reserved word: if
      6: NUM, val= 0
      6: <
      6: ID, name = x
      6: reserved word: then
   7:  fact := 1;
      7: ID, name= fact
      7: :=
      7: NUM, val=1
      7: ;
   8:    repeat
      8: reserved word repeat
   9:       fact := fact * x;
      9: ID, name= fact
      9: :=
      9: ID, name= fact
      9: *
      9: ID, name= x
      9: ;
  10:     x := x - 1
     10: ID, name= x
     10: :=
     10: ID, name = x
     10: -
     10: NUM, val= 1
  11:  until x = 0;
     11: reserved word: until
     11: ID, name= x
     11: =
     11: NUM, val= 0
     11: ;
  12:  write fact {saída do fatorial de x}
     12: reserved word: write
     12: ID, name= fact
  13: end
     13: reserved word: end
     14: EOF
```

A linguagem TINY e o gerador FLEX

Exemplo:

	letra	dígito	outro	Aceitação
1	2 V			F
2	2 V	2 V	[3] F	F
3				V





A linguagem TINY e o gerador FLEX

- O processo de construção de um analisador léxico pode ser automatizado;
- Existem vários geradores de analisadores léxicos disponíveis gratuitamente, que ajudam muito no desenvolvimento de um compilador;
- Um gerador muito conhecido é o *Flex (Fast Lex)*
  - Distribuído como parte do pacote de compilação *GNU* (produzido pela *Free Software Foundation*).

A linguagem TINY e o gerador FLEX

Formato do arquivo de entrada Flex

- O arquivo de entrada é composto por três partes
  - Definições;
  - Regras;
  - Rotinas do usuário.
- As seções são separadas por dois sinais de porcentagem, que aparecem juntos em linhas separadas, iniciando na primeira coluna.

A linguagem TINY e o gerador FLEX

Formato do arquivo de entrada Flex

```
%{
    qualquer código C a ser inserido externamente a
    qualquer função        opcional
%}
definições regulares        opcional
%%
expressões regulares seguidas do código C a ser executado quando houver
casamento com a expressão regular correspondente
%%
código C para rotinas auxiliares ativadas pela segunda seção, pode também
conter um programa principal                    opcional
```

A linguagem TINY e o gerador FLEX

Convenções de metacaracteres em Lex

Tabela 2.2 Convenções de metacaracteres em Lex

Padrão	Significado
`a`	caractere *a*
`"a"`	caractere *a*, mesmo se *a* for um metacaractere
`\a`	caractere *a* se *a* for um metacaractere
`a*`	zero ou mais repetições de *a*
`a+`	uma ou mais repetições de *a*
`a?`	um *a* opcional
`a|b`	*a* ou *b*
`(a)`	*a* propriamente dito
`[abc]`	qualquer caractere entre *a*, *b* e *c*
`[a-d]`	qualquer caractere entre *a*, *b*, *c* e *d*
`[^ab]`	qualquer caractere, exceto *a* ou *b*
`.`	qualquer caractere, exceto mudança de linha
`{xxx}`	a expressão regular representada pelo nome *xxx*

(pela interpretação ANSI –C)

A linguagem TINY e o gerador FLEX

Exemplo de utilização do Flex (arquivo de entrada para o Flex)

```
/*
 * Description: Count the number of characters and the number of lines from standard input
 * */
%{
int num_lines = 0, num_chars = 0;
%}

%%
\n   ++num_lines; ++num_chars;
.    ++num_chars;
fim  return 0;
%%

main()
{
yylex();
printf("# of lines = %d, # of chars = %d\n", num_lines, num_chars);
}
```

A linguagem TINY e o gerador FLEX

Exemplo de utilização do Flex

- Reconhecimento de cadeias que atendam a especificação de números inteiros nos formatos:
  - Decimal;
  - Octal;
  - Hexadecimal;
  - Binário.

- Assim, entradas como “abc 10 def 017 ghi 0xAF0” têm como saída “abc Dec def Oct ghi Hex”.

A linguagem TINY e o gerador FLEX

Exemplo de utilização do Flex

```
DIGIT [0-9]
%%
[1-9]{DIGIT}*                printf("DEC");
0[0-7]*                      printf("OCT");
0x[0-9A-Fa-f]+               printf("HEX");
0b[01]+                      printf("BIN");
<<EOF>>                      return 0;
%%
int main(int argc, char *argv[])
{
  FILE *f_in;

  if (argc == 2)
  {
    if(f_in = fopen(argv[1],"r"))  yyin = f_in;
    else  perror(argv[0]);
  }
  else  yyin = stdin;

  yylex();
  return(0);
}
```

- Arquivo de entrada (programa) a ser lido pelo gerador de analisador léxico (Flex*)*
- Este arquivo deve ter extensão .l
- No exemplo, o nome do arquivo é *exemplo.l*

A linguagem TINY e o gerador FLEX

Nomes internos utilizados por Flex

Tabela 2.3 Alguns nomes internos Lex

Nome Interno Lex	Significado/Utilização
`lex.yy.c or lexyy.c`	Arquivo de saída Lex
`yylex`	Rotina de varredura Lex
`yytext`	Cadeia casou com ação corrente
`yyin`	Entrada Lex (padrão: `stdin`)
`yyout`	Saída Lex (padrão: `stdout`)
`input`	Rotina de entrada com reservatório Lex
`ECHO`	Ação básica Lex (imprime `yytext` em `yyout`)

A documentação completa de *FLEX* está disponível em http://flex.sourceforge.net/manual/

A linguagem TINY e o gerador FLEX

Exemplo de utilização do Flex

- Ao digitar *flex exemplo.l*, o Flex gerará o analisador léxico, com o seguinte nome padrão *lex.yy.c*;

- *lex.yy.c* implementa os procedimentos do analisador léxico, mas ainda precisa ser compilado e ligado com a biblioteca *libfl*

  gcc –o exec lex.yy.c -lfl

A linguagem TINY e o gerador FLEX

Exemplo de utilização do Flex

Podemos executar *exec* com ou sem um arquivo texto de entrada

- Sem arquivo de entrada

  exec fará a leitura de caracteres de entrada via teclado

  (yyin = stdin)

- Com arquivo de entrada

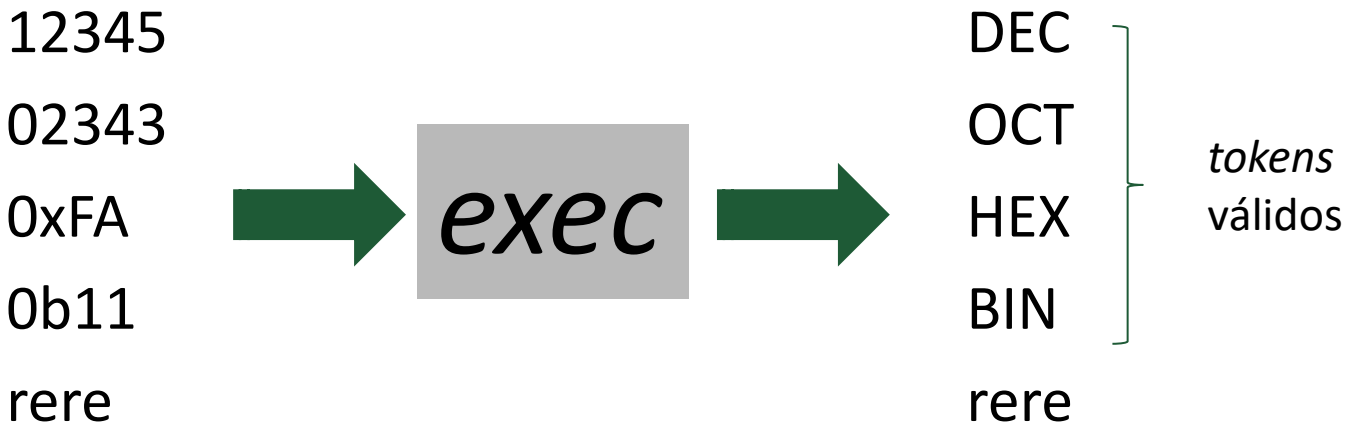
  exec fará a leitura de caracteres de entrada via arquivo

  (yyin = f_in)

A linguagem TINY e o gerador FLEX

Exemplo de utilização do Flex

Considere um arquivo de entrada que contenha as seguintes cadeias de caracteres:



A linguagem TINY e o gerador FLEX

1. Download do Flex: http://gnuwin32.sourceforge.net/packages.html;
2. Instalar o Flex;
3. Adicionar a pasta caminho_do_flex/bin ao Path nas variáveis de ambiente do seu usuário no computador;
4. Salvar o arquivo de entrada a ser oferecido ao Flex com extensão .l;

A linguagem TINY e o gerador FLEX

5. Pelo prompt de comando, executar o comando *flex nome_do_arquivo.l* obtendo o lex.yy.c;

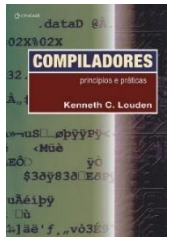
6. Gerar o executável pelo comando *gcc –o exec lex.yy.c -lfl* no prompt de comando ou Cygwin, ou ainda pelo DevC++ ou Codeblocks;

7. Executar o arquivo exec obtido no passo anterior e verificar o reconhecimento das expressões regulares definidas.

A linguagem TINY e o gerador FLEX

Bibliografia consultada

RICARTE, I. **Introdução à Compilação.** Rio de Janeiro: Editora Campus/Elsevier, 2008.



LOUDEN, K. C. **Compiladores: princípios e práticas.** São Paulo: Pioneira Thompson Learning, 2004.

Agradeço ao Professor Luiz Eduardo G. Martins pelo material disponibilizado.